AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Dokumentacja projektu

Temat:
Travel Agency App
Aplikacja do dokonywania rezerwacji na wycieczki dla biura podróży

Przedmiot
Bazy danych

Prowadzący
dr inż. Robert Marcjan

Skład grupy:
Jakub Nowobilski
Piotr Kasprzyk

Krótka charakterystyka opracowanego systemu - wstępne założenia i funkcjonalności (uwaga: elementy przekreślone nie zostały ostatecznie zrealizowane**):**
Opracowano system (aplikację Travel Agency App) do dokonywania rezerwacji na wycieczki dla biura podróży zapewniająca podstawową funkcjonalność rezerwacji danej wycieczki.
Rezerwacje na wycieczki będą mogły mieć status: “nowa” (N), “opłacona” (P), “odwołana” (C).
Z aplikacji korzystać będą dwa rodzaje użytkowników pod względem roli i uprawnień (zwykły użytkownik - R oraz administrator - A).
Jeśli chodzi o korzystanie z aplikacji:
- pracowniki biura podróży - będzie miał np. możliwość dodawania nowych wycieczek, czy wycofywania wycieczek z oferty biura podróży;
- użytkownik - będzie miał m.in. możliwość dokonywania rezerwacji na wybraną wycieczkę lub dokonywania płatności za wycieczkę.

~~W/w użytkowników aplikacji przechowujemy w Fire Base~~, a dane uczestników wycieczek (nie korzystają oni z aplikacji) są przechowywane w odpowiedniej tabela w relacyjnej bazie danych.

Dodatkowe funkcjonalności:
- pozostawianie recenzji wycieczek dostępnych w ofercie firmy przez uczestników
- dodawanie ocen do wycieczek

Stos technologiczny (uwaga: technologie przekreślone nie zostały ostatecznie wykorzystane**):**

Technologia / narzędzie	**Zastosowanie**
PostgreSQL	serwer bazodanowy
~~FireBase~~	~~usługa uwierzytelniająca oraz autentykująca użytkowników~~
Java + Spring Boot	warstwa backend aplikacji
~~Angular~~	~~warstwa frontend aplikacji~~

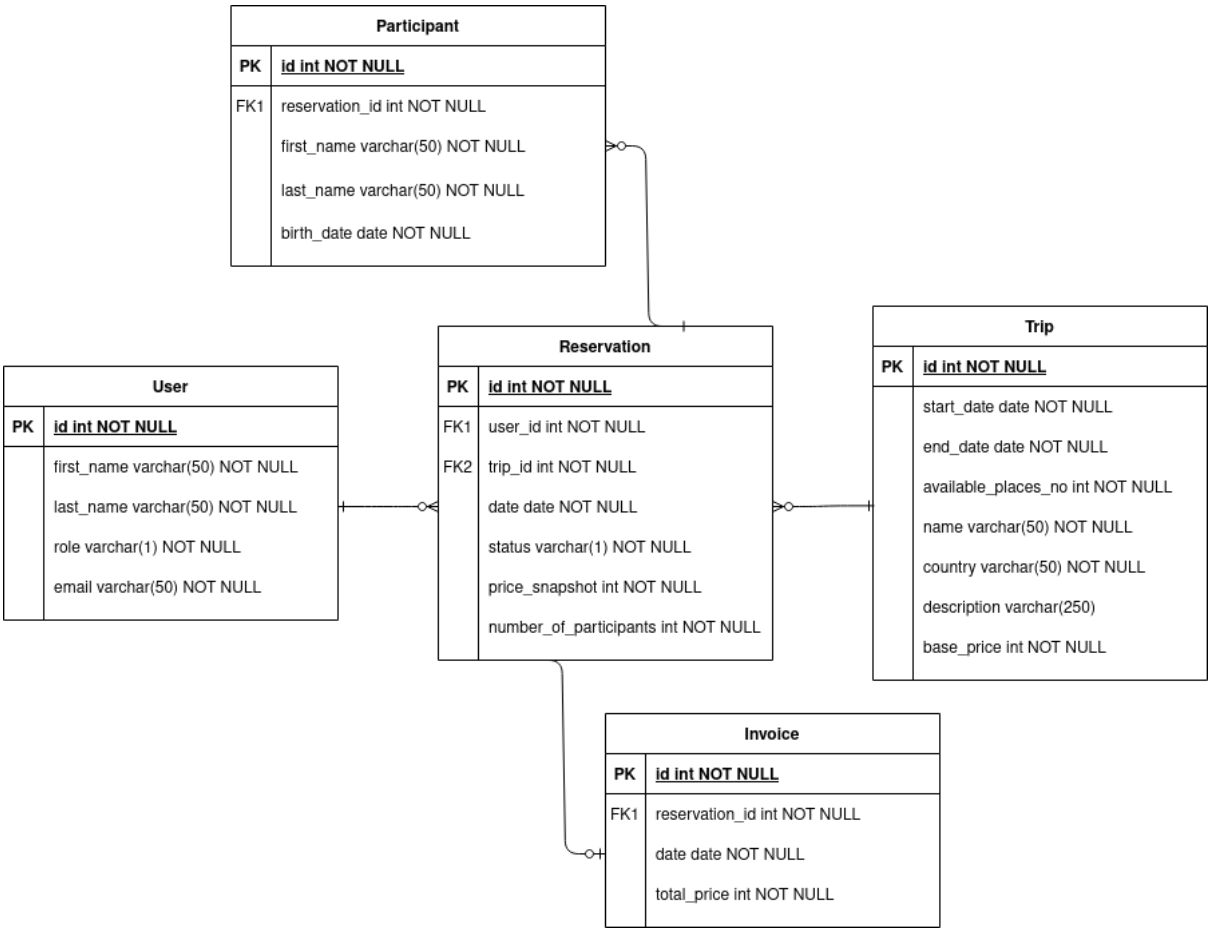
Uwaga: Ponad zaplanowane na początku części systemu (aplikację backendową w Spring Boot z Hibernate wykorzystującą serwer bazodanowy PostgreSQL) w ramach projektu zrealizowano również aplikację generatora danych (link do repozytorium został zawarty poniżej). Została ona napisana w NodeJs z wykorzystaniem biblioteki axios zawierającej zbiory danych (m.in. imiona, nazwiska, nazwy krajów itp.).

Link do repozytorium na github z systemem (aplikacją backendową):
https://github.com/kaspiotr/TravelAgencyApp

Link do repozytorium na github z generatorem danych:
https://github.com/JakubNowobilski/TravelAgencyApp-DataGenerator

Link do REST API aplikacji hostowanej na Heroku:
https://travel-agency-app-heroku.herokuapp.com/swagger-ui/index.html?url=/v3/api-docs#/

Schemat bazy danych zastosowany w projekcie



Kaskady

W aplikacji zastosowano tzw. kaskady, tzn. encje zawierające (ang. owning entities), które w razie usunięcia usuwają także wszystkie swoje zależności (wszystkie encje, które się w nich zawierają). Przykładowo w tej aplikacji przy usuwaniu rezerwacji (encja **Reservation**) usuwana jest faktura (encja **Invoice**) z nią powiązana oraz uczestnicy (encja **Participant**) wymienieni w danej rezerwacji.

Funkcjonalności aplikacji w ramach danej tabeli

1. Tabela **User**
Przechowuje dane o użytkownikach aplikacji. Encja **User** jest powiązana z encją **Reservation** relacją jeden - do - wielu.
- Pole **id**: int jest kluczem głównym w tabeli generowanym automatycznie i nie może przyjąć wartości NULL
- Pole **first_name**: varchar(50) przechowuje imię użytkownika i nie może być NULLem
- Pole **last_name**: varchar(50) przechowuje nazwisko użytkownika i nie może być NULL
- Pole **role**: varchar(1) określa rolę użytkownika: administrator - A, zwykły użytkownik - R
- Pole **email**: varchar(50) przechowuje informację o adresie e-mail użytkownika

Funkcjonalności systemu w ramach tabeli **User**:
- tworzenie jednego użytkownika aplikacji;
- pobieranie informacji na temat wybranego użytkownika aplikacji po jego id;
- pobieranie informacji o wszystkich użytkownikach;
- usuwanie jednego użytkownika podając jego id;
- modyfikowanie jednego użytkownika podając jego id - modyfikować można dowolną kombinację poniższych atrybutów opisujących użytkownika:
  - first_name
  - last_name
  - mail (musi być unikalny)
- filtrowanie użytkowników - użytkowników można filtrować wykorzystując dowolną kombinację poniższych atrybutów:
  - firstName
  - lastName
- pobieranie wszystkich skojarzonych rezerwacji (endpoint: users/{id}/reservations).

2. Tabela **Trip**
Przechowuje dane o wycieczkach. Encja **Trip** jest w powiązana relacją jeden - do - wielu z encją **Reservation**.
- Pole **id**: int jest kluczem głównym w tabeli generowanym automatycznie i nie może przyjąć wartości NULL
- Pole **start_date**: date określa dzień rozpoczęcia wycieczki i nie może przyjąć wartości NULL,
- Pole **end_date**: date określa dzień zakończenia wycieczki i nie może przyjąć wartości NULL,
- Pole **available_places_no**: int określa liczbę wolnych miejsc na daną wycieczkę i nie może być NULL,
- Pole **name**: varchar(50) przechowuje nazwę wycieczki i nie może być puste,
- Pole **country**: varchar(50) przechowuje informacje o kraju, w którym odbywa się wycieczka i nie może przyjąć wartości NULL,
- Pole **description**: varchar(250) zawiera opis i dodatkowe informacje na temat wycieczki, może ono przyjmować wartość NULL,
- Pole **base_price**: int przechowuje informacje o cenie podstawowej wycieczki i może ulec zmianie (faktyczna cena jest zapisywana w momencie utworzenia rezerwacji do pola **price_snapshot** w tabeli **Reservation**).

Funkcjonalności systemu w ramach tabeli **Trip**:
- tworzenie jednej wycieczki;
- pobieranie informacji na temat wybranej wycieczki po jej id;
- pobieranie informacji o wszystkich wycieczkach;
- usuwanie jednej wycieczki podając jej id (endpoint dostępny jedynie dla administratora / pracownika biura podróży; przy usuwaniu wycieczki administratora systemu informuje o tym fakcie uczestników wycieczki; przy usuwaniu wycieczki usuwane są także wszystkie powiązane z nią rezerwacje);
- filtrowanie wycieczek - wycieczki można filtrować wykorzystując dowolną kombinację poniższych atrybutów:
  - country;
  - priceFrom;
  - priceTo;
  - placesFrom;
  - placesTo;
  - duration;
  - dateFrom;
  - dateTo;
- pobieranie wszystkich skojarzonych rezerwacji (enpoint: trips/{id}/reservations);
- przeglądanie listy wycieczek - do liczenia liczby dostępnych miejsc (endpointy: /trips/withPlaces/{id} oraz /trips/withPlaces, które zwracają odpowiednio obiekty ExtendedTrip oraz List<ExtendedTrip>, gdzie ExtendedTrip to obiekt trip wraz z dodatkowym polem availablePlacesNumber);

3. Tabela **Reservation**
Przechowuje dane o rezerwacjach danej osoby na daną wycieczkę. W istocie stanowi tabelę łącznikową między tabelami **User** a **Trip**. Encja tej tabeli nie może zatem istnieć bez odpowiednich encji powyższych tabel. Posiada opcjonalną relację jeden - do - jeden z tabelą **Invoice** oraz jeden - do - wielu z tabelą **Participant**. Encja **Reservation** jest tzw. encją zawierającą (ang. owning entity), dlatego przy jej usunięciu usuwane są także faktury (encja **Invoice**) z nią powiązane oraz uczestnicy (encja **Participant**) wymienieni w danej rezerwacji.
- Pole **id**: int jest kluczem głównym w tabeli generowanym automatycznie i nie może przyjąć wartości NULL
- Pole **user_id**: int klucz obcy powiązany z kluczem głównym w tabeli **User**
- Pole **trip_id**: int klucz obcy powiązany z kluczem głównym w tabeli **Trip**
- Pole **date**: date odpowiada dacie utworzenia rezerwacji.
- Pole **status**: varchar(1) odpowiada statusowi rezerwacji (**C - Cancelled, P - Paid, N - New)**
- Pole **price_snapshot**: int odpowiada zrzutowi ceny wycieczki w momencie utworzenia rezerwacji
- Pole **number_of_participants**: int odpowiada deklarowanej liczbie uczestników w ramach rezerwacji.

Funkcjonalności systemu w ramach tabeli **Reservation**:
- tworzenie nowej rezerwacji - stworzenie nowej rezerwacji możliwe jest tylko na wycieczki, które się odbędą w przyszłości oraz pod warunkiem, że dostępna jest odpowiednia liczba dostępnych miejsc;
- modyfikowanie rezerwacji - dla nowej wycieczki możliwa jest zmiana zadeklarowanej liczby miejsc. Możliwe jest również modyfikowanie statusu: **N -> C, P / P -> C**
- zmiana statusu na **Paid** możliwa jest tylko jeżeli faktyczna liczba dodanych uczestników odpowiada liczbie zadeklarowanych uczestników;
- zmiana zadeklarowanej liczby miejsc jest możliwa jedynie wtedy gdy miejsc wolnych na daną wycieczkę jest odpowiednio wiele (przy zwiększaniu liczby miejsc na daną wycieczkę) lub jeśli liczba aktualnych uczestników będzie nie większa niż nowa deklarowana liczba miejsc (przy zmniejszaniu liczby miejsc na daną wycieczkę);
- automatyczne tworzenie faktury (**Invoice**) w momencie zmiany statusu na **Paid**;
- pobieranie rezerwacji po jej id;
- pobieranie wszystkich rezerwacji;
- pobieranie skojarzonej faktury;
- pobieranie wszystkich skojarzonych uczestników;
- przeglądanie listy wycieczek;
- rezygnacja z rezerwacji (jest realizowane przez zmianę jej statusu na **C - Cancelled**; endpoint: changeReservationStatus(reservationId:Long, status: String) /reservations/{id}/changeStatus?status=<status>)

4. Tabela **Invoice**
Przechowuje dane o fakturach stworzonych dla danych rezerwacji. Odpowiada dokładnie jednej rezerwacji. Encja tabeli **Invoice** nie może istnieć bez tabeli **Reservation.**
- Pole **id**: int jest kluczem głównym w tabeli generowanym automatycznie i nie może przyjąć wartości NULL;
- Pole **reservation_id**: int jest kluczem obcym powiązanym z kluczem głównym w tabeli **Reservation** i nie może przyjmować wartości NULL;
- Pole **date**: date jest generowane automatycznie i odpowiada dacie utworzenia faktury, nie może być puste;
- Pole **total_price**: int jest zrzutem sumarycznej ceny w momencie opłacenia zamówienia i nie może przyjmować wartości NULL.

Funkcjonalności systemu w ramach tabeli **Invoice**:
- pobieranie faktury po jej id;
- pobieranie wszystkich faktur.

5. Tabela **Participant**
Przechowuje dane o uczestnikach w ramach danej rezerwacji. Uczestnika od użytkownika (**User**) odróżnia to, że drugi jest użytkownikiem systemu, w którym tworzone są rezerwacje, a użytkownicy są do nich przypisywani. Dany uczestnik należy dokładnie do jednej rezerwacji. Encja tabeli **Participant** nie może istnieć bez tabeli **Reservation.** Encja **Participant** jest powiązana relacją wiele - do - jeden z encją **Reservation**.
- Pole **id**: int jest kluczem głównym w tabeli generowanym automatycznie i nie może przyjąć wartości NULL;
- Pole **reservation_id**: int klucz obcy powiązany z kluczem głównym w tabeli **Reservation**;
- Pole **first_name**: varchar(50) przechowuje imię uczestnika wycieczki (nie może być NULLem);
- Pole **last_name**: varchar(50) przechowuje nazwisko uczestnika wycieczki (nie może być NULLem);
- Pole **birth_date**: date przechowuje informacje o dacie urodzenia uczestnika wycieczki (także nie może przyjmować wartości NULL).

Funkcjonalności systemu w ramach tabeli **Participant**:
- tworzenie nowego uczestnika - w ramach istniejącej rezerwacji;
- modyfikowanie użytkownika;
- usuwanie danego użytkownika - usunięcie z danej rezerwacji;
- pobranie skojarzonej rezerwacji;
- pobranie skojarzonego użytkownika;
- pobranie skojarzonej wycieczki;
- pobieranie użytkownika;
- pobieranie wszystkich użytkowników.

Uwagi, problemy i rozwiązania:

W trakcie prac na system napotkaliśmy m.in. na problem z rekursywnym odpytywaniem przy parsowaniu danych do postaci JSON. Dlatego też zastosowaliśmy rozwiązanie oparte o konstruktory kopiujące. Poniższe zrzuty stanowią przykład napotkanego problemu: